

# HS MSG C Source Library v1.0.1

*Message Oriented Middleware Library*

## User Manual

16 February 2012

Rev 1.1

### Table of Contents

1 Introduction.....	3
1.1 General.....	3
1.2 Limitations in the evaluation version.....	3
1.3 How to use the library.....	3
2 HS MSG API.....	4
2.1 Functions.....	4
2.1.1 HsMsgInit.....	4
2.1.2 HsMsgCleanUp.....	4
2.1.3 HsMsgCreateInbox.....	5
2.1.4 HsMsgDestroyInbox.....	6
2.1.5 HsMsgAddQueue.....	6
2.1.6 HsMsgRemoveQueue.....	7
2.1.7 HsMsgTick.....	7
2.1.8 HsMsgReadQueue.....	8
2.1.9 HsMsgPostMessage.....	9
2.1.10 HsMsgSendMessage.....	10
2.2 Structures.....	12
2.2.1 hsmsg_message_t.....	12
2.3 Definitions.....	13
2.3.1 Return Codes.....	13
3 Sample Applications.....	15
3.1 hsmsg_chat.....	15
3.1.1 Purpose.....	15
3.1.2 Syntax.....	15
3.1.3 Usage.....	15
3.2 sendfile.....	16
3.2.1 Purpose.....	16
3.2.2 Syntax.....	16
3.2.3 Usage.....	16
3.3 recvfile.....	17
3.3.1 Purpose.....	17
3.3.2 Syntax.....	17



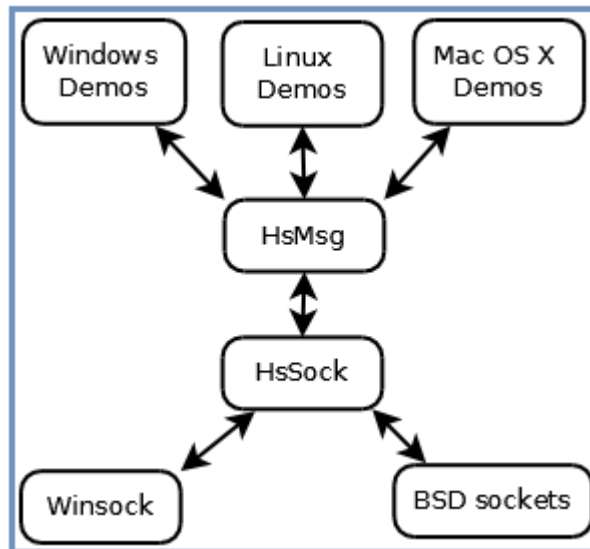
# 1 Introduction

## 1.1 General

HS MSG is a C messaging library (message oriented middleware) for communication between threads of the same process and between different processes on the same or on different computers. HS MSG C messaging SDK runs on Windows, Linux, MAC OSX, iPhone iOS.

HS MSG is intended to be used from C/C++ applications.

The terms of HS MSG source code license provide for the royalty free use, derivative works, use within and outside your company and re-distribution as part of your products.



## 1.2 Limitations in the evaluation version

The evaluation version of HS MSG Library has the following limitations

- Windows version displays a message box requiring user clicking OK before initialization continues.
- Linux and MAX OS X versions display Hillstone Software copyright information at startup.
- Total number of messages allowed to send is limited to 100. After that the send function returns failure code -18. The application must re-start to reset this count.

## 1.3 How to use the library

First user application calls HsMsgInit() initialization function to initialize the HsMsg library

Then the user application can create a message "Inbox" for receiving messages from other peers. A parameter to HsMsgCreateInbox function is an IP port number on which HsMsg shall receive messages

After that the application call create one or more message queues associated with this inbox, calling HsMsgAddQueue(). Each queue is given a string name. The remote (or local) entities will send messages to the inbox and the specific queue name

In order to allow HsMsg to drive internal timing, reception and processing of data from underlying transport layer, the application must periodically call HsMsgTick()

Once at least one queue has been created, an applicatin can poll the queue for messages by periodically calling HsMsgReadQueue, specifying the inbox handle, the queue handle and a pointer to the message structure

To send messages without requiring the result, the application calls HsMsgPostMessage(), specifying the inbox handle on which the message will go out, the destination address, which consists of the remote IP

address and port and the destination queue name. HsMsgPostMessage() returns immediately and the messages sending is handled automatically by HsMsg in the background

To send messages synchronously the application calls HsMsgSendMessage(), this function does not return until the message has been fully delivered to the destination queue or a permanent (unrecoverable) error occurred.

HsMsg library internally implements reliable transmission of the message content over UDP, dealing with message segmentation, re-assembly, multiplexing and de-multiplexing, re-transmissions, etc.

When finished working with the library (typically before exit) the application calls HsMsgCleanUp() to de-initialize the library

## 2 HS MSG API

To call HS MSG API functions include the following header file:

hsmsg\_if.h

### 2.1 Functions

#### 2.1.1 HsMsgInit

Initializes HS MSG library. Call this function once at program initialization, before any other functions are called.

##### Syntax:

```
int HsMsgInit(void);
```

##### Parameters:

None.

##### Return values:

If the function succeeds the return value is 0. If the function fails, the return value is a negative integer, one of the values specified below:

Return Code	Description
-10	Socket layer initialization failed

##### Remarks:

This function must be called before any other functions of the library are called. Call this function once at the startup of your application. On application exit, call HsMsgCleanUp to de-allocate the resources used by HS MSG library.

#### 2.1.2 HsMsgCleanUp

De-Initializes HS MSG library. Call this function once at your program exit.

##### Syntax:

```
void HsMsgCleanUp(void);
```

**Parameters:**

none.

**Return values:**

none.

**Remarks:**

This function must be called before you exit your application to cleanup and release all resources used by HS MSG library.

### 2.1.3 HsMsgCreateInbox

Creates a message inbox for receiving incoming messages, The inbox id that is returned by this function is also necessary for sending messages.

**Syntax:**

```
int HsMsgCreateInbox(unsigned short *pPort);
```

**Parameters:**

\*pPort pointer to unsigned short integer containing the value of IP port to use for the inbox. This IP port number shall be used to create a UDP socket which is then used both to send and receive messages.

To create an inbox on a specific IP port, set the value in the range from 1 to 65535. To have the underlying socket layer select the port number automatically, set the value to 0. On return, the \*pPort is set to the value of IP port actually used.

**Return values:**

If the function succeeds the return value is a non-negative inbox id. If the function fails, the return value is a negative integer, one of the values specified below:

Return Code	Description
-6	Library not initialized
-1	Invalid parameters
-7	Maximum number of inboxes reached, cannot allocate a new inbox
-8	Failed to open a socket for the inbox
-9	Failed to bind the socket for the inbox

**Remarks:**

Use this function to create a message inbox. A message inbox is associated with a local IP socket port number. One application can create more than one inbox. The current version of HS MSG supports 64 inboxes. The inbox id returned by this function is necessary for receiving and sending messages.

To create an inbox on a specific IP port, set the value in the range from 1 to 65535. To have the underlying socket layer select the port number automatically, set the value to 0. On return, the \*pPort is set to the value of IP port actually used.

## 2.1.4 HsMsgDestroyInbox

Destroys the specified inbox.

### Syntax:

```
int HsMsgDestroyInbox(int inbox_id);
```

### Parameters:

inbox\_id - integer Inbox id obtained by calling HsMsgCreateInbox().

### Return values:

If the function succeeds the return value is 0. If the function fails, the return value is a negative integer, one of the values specified below:

Return Code	Description
-6	Library not initialized
-1	Invalid parameters
-2	Inbox not allocated

### Remarks:

Use this function to create a message inbox. A message inbox is associated with a local IP socket port number. One application can create more than one inbox. The current version of HS MSG supports 64 inboxes. The inbox id returned by this function is necessary for receiving and sending messages.

To create an inbox on a specific IP port, set the value in the range from 1 to 65535. To have the underlying socket layer select the port number automatically, set the value to 0. On return, the \*pPort is set to the value of IP port actually used.

## 2.1.5 HsMsgAddQueue

Adds a messages queue to the previously created inbox.

### Syntax:

```
int HsMsgAddQueue(int inbox_id, char *local_queue_name);
```

### Parameters:

inbox\_id - integer Inbox id obtained by calling HsMsgCreateInbox().

\*local\_queue\_name – pointer to NULL terminated ASCII string that is the name of the queue. The string buffer is in multibyte (non unicode) format. The maximum length of the string buffer is 32 bytes including the terminating zero byte.

### Return values:

If the function succeeds the return value is a non-negative queue id. If the function fails, the return value is a negative integer, one of the values specified below:

Return Code	Description
-6	Library not initialized

-1	Invalid parameters
-11	Queue name already exists in the list of queues of the specified inbox
-12	Failed to allocate a new queue, maximum number of queues per inbox reached

**Remarks:**

Use this function to add a local message queue to the previously created inbox. Multiple local message queues can be added to the same inbox. Current version of HS MSG supports a maximum of 10 queues per inbox. A queue must have a unique queue name in the context of one inbox. A queue is used for reception of messages. The message sender specifies the destination queue name as one of the components of the message destination address.

## 2.1.6 HsMsgRemoveQueue

Removes the message queue from the specified inbox.

**Syntax:**

```
int HsMsgRemoveQueue(int inbox_id, int queue_id);
```

**Parameters:**

inbox\_id - integer Inbox id obtained by calling HsMsgCreateInbox().

queue\_id – integer queue id obtained by calling HsMsgAddQueue()

**Return values:**

If the function succeeds the return value is 0. If the function fails it returns one of the values listed below:

Return Code	Description
-6	Library not initialized
-1	Invalid parameters
-2	Inbox not allocated
-13	Queue not allocated

**Remarks:**

Use this function to remove / de-allocate the message queue.

## 2.1.7 HsMsgTick

HS MSG tick function. This function must be called periodically as often as possible, by the user application.

**Syntax:**

```
void HsMsgTick(int inbox_id);
```

**Parameters:**

inbox\_id - integer Inbox id obtained by calling HsMsgCreateInbox().

### Return values:

none

### Remarks:

This function must be called periodically by the user application, as often as possible. This function drives internal timers and network socket level operations and is necessary for both message transmission and reception through the specified inbox.

## 2.1.8 HsMsgReadQueue

Reads one message from the specified message queue, if a message is available for reading.

### Syntax:

```
int HsMsgReadQueue(int inbox_id, int queue_id, hsmsg_message_t *pMsg);
```

### Parameters:

inbox\_id - integer Inbox id obtained by calling HsMsgCreateInbox().

queue\_id – integer queue id obtained by calling HsMsgAddQueue()

\*pMsg – pointer to the message structure where the received message is copied to.

Refer to section 2.2 for the definition of the message structure hsmsg\_message\_t

### Return values:

If the function succeeds, the return value is 0 and the received message is copied into the structure pointed to by \*pMsg. Refer to section 2.2 for the definition of the message structure hsmsg\_message\_t.

If there are no messages available in the specified queue, the return value is -16.

If the function fails for another reason, the return value is one of the values listed below:

Return Code	Description
-6	Library not initialized
-1	Invalid parameters
-2	Inbox not allocated
-13	Queue not allocated

### Remarks:

Use this function to receive messages. This function must be called periodically, as often as possible, for each message queue associated with the inbox in order to read the messages from the queue.

This function is non-blocking and returns immediately. Return code -16 indicates that the queue is empty. A return code of 0 indicates success - the message has been received and copied into \*pMsg.

For detailed definition of the message structure fields, please refer to section 2.2

A message received has a number of fixed fields and may or may not have an associated data buffer. The fixed fields include the buffer length buflen and the buffer pointer \*pBuf. If a message contains the data buffer, the buflen field is non zero and is set to the length of the data and the \*pBuf points to the buffer data. If a message contains no buffer data, the buflen is zero and the \*pBuf is NULL.

The maximum length of the data buffer is 65535 bytes.

If the message contains the data buffer (pBuf is not NULL), the user application is responsible for freeing the memory pointed to by \*pBuf when it no longer needs it by calling free() function from C standard library.

## 2.1.9 HsMsgPostMessage

Sends a message to the destination and returns immediately without waiting for the result.

### Syntax:

```
int HsMsgSendMessage(  
    int            inbox_id,  
    char           *destination_address,  
    char           *destination_queue_name,  
    hsmsg_message_t *pMsg);
```

### Parameters:

**inbox\_id** - integer inbox id obtained by calling HsMsgCreateInbox(). This is the local inbox id through which the message will be sent.

**\*destination\_address** – pointer to zero terminated destination address where the message should be sent. The address format is as follows:

“hostname:port”, where hostname can be either a DNS name of the destination or IP address in dotted format (for example “192.168.1.1”) and port is the IP port of the destination. IP port is the same as the destination used when calling HsMsgCreateInbox() function.

Example of full address:

“192.168.1.1:1234”

The string pointed to by \*destination\_address must be zero terminated and must be in multi-byte (non unicode) format.

**\*destination\_queue\_name** – name of the destination queue associated with the remote inbox where the message is to be delivered. The name should match one of the names the destination used to create a queue by calling HsMsgAddQueue().

The string pointed to by \*destination\_queue\_name must be zero terminated and must be in multi-byte (non unicode) format.

The maximum length of the destination queue name including the zero terminating byte is 32 bytes.

**\*pMsg** – pointer to the message structure that contains the message to send. For the definition of the message structure, please refer to section 2.2

### Return values:

If the function succeeds, the return value is a non negative integer. If the function fails, the return value is a negative integer, one of the following:

Return Code	Description
-6	Library not initialized
-1	Invalid parameters

-2	Inbox not allocated
-3	Out of session contexts. A maximum of 100 message transfer sessions can be in progress simultaneously in the current version of HS MSG.
-5	Failed to generate a source ID for the message session
-4	Memory allocation error – out of memory
-18	If running the evaluation version, number of messages allowed to send exceeded.

### Remarks:

Use this function to send a message to the destination without waiting for the result. This function is non-blocking, it returns immediately both in case of success and failure. The success return code indicates that the message transfer to the destination has been initiated successfully.

Even though this function does not wait for the delivery of the message into the destination queue on the remote inbox, in the background HS MSG will always attempt to reliably deliver the message to the destination. This is achieved by internal transport protocol over UDP using sequence numbers, acknowledgements, timers and re-transmissions.

For the definition of the message structure, please refer to section 2.2. The message structure has two fields that relate to the data buffer. Field `*pBuf` is a pointer to the data buffer linked to the message. Field `buflen` is the length of the data buffer linked to the message.

To send a message without a data buffer set `*pBuf` to NULL and the `buflen` to 0.

To send a message with a data buffer, set `*pBuf` to the address of the buffer and `buflen` to the length of the buffer in bytes.

If `pBuf` is not NULL and `buflen` is not 0, HS MSG copies the data buffer into internal buffers. The application can dispose of the buffer (if it is not statically allocated) on return from this function.

The maximum length of the data buffer is 65535 bytes.

As explained in the specification of function `HsMsgTick()`, it is necessary to call `HsMsgTick()` periodically as often as possible in order for messages to be delivered to destination and for incoming messages to be received into the message queues.

Messages can be sent both to remote destinations located on different hosts or to message queues located on the same host as the sender. In order to send to a local queue, use local loopback address (127.0.0.1), local inbox port in the `destination_address` string, for example "127.0.0.1:1234" and specify the desired local queue name in the `destination_queue_name` parameter.

## 2.1.10 HsMsgSendMessage

Sends a message to the destination and waits for the message to be delivered into the destination queue.

### Syntax:

```
int HsMsgSendMessage(
    int                inbox_id,
    char               *destination_address,
    char               *destination_queue_name,
    hsmmsg_message_t  *pMsg)
```

### Parameters:

`inbox_id` - integer inbox id obtained by calling `HsMsgCreateInbox()`. This is the local inbox id through which the message will be sent.

`*destination_address` – pointer to zero terminated destination address where the message should be sent.

The address format is as follows:

“hostname:port”, where hostname can be either a DNS name of the destination or IP address in dotted format (for example “192.168.1.1”) and port is the IP port of the destination. IP port is the same as the destination used when calling HsMsgCreateInbox() function.

Example of full address:

“192.168.1.1:1234”

The string pointed to by \*destination\_address must be zero terminated and must be in multi-byte (non unicode) format.

\*destination\_queue\_name – name of the destination queue associated with the remote inbox where the message is to be delivered. The name should match one of the names the destination used to create a queue by calling HsMsgAddQueue().

The string pointed to by \*destination\_queue\_name must be zero terminated and must be in multi-byte (non unicode) format.

The maximum length of the destination queue name including the zero terminating byte is 32 bytes.

\*pMsg – pointer to the message structure that contains the message to send. For the definition of the message structure, please refer to section 2.2

### Return values:

If the function succeeds, the return value is 0 – the message has been delivered into the destination queue.

If the function fails, the return value is a negative integer, one of the following:

Return Code	Description
-6	Library not initialized
-1	Invalid parameters
-2	Inbox not allocated
-3	Out of session contexts. A maximum of 100 message transfer sessions can be in progress simultaneously in the current version of HS MSG.
-5	Failed to generate a source ID for the message session
-4	Memory allocation error – out of memory
-14	Destination queue not found
-15	Destination queue is full
-17	Timeout
-18	If running the evaluation version, number of messages allowed to send exceeded.

### Remarks:

Use this function to send a message to the destination and wait for the result. This function is a synchronous blocking call, it does not return until a message has been delivered or one of error conditions occur.

HS MSG will reliably deliver the message to the destination. This is achieved by internal transport protocol over UDP using sequence numbers, acknowledgements, timers and re-transmissions.

For the definition of the message structure, please refer to section 2.2. The message structure has two fields that relate to the data buffer. Field \*pBuf is a pointer to the data buffer linked to the message. Field buflen is the length of the data buffer linked to the message.

To send a message without a data buffer set \*pBuf to NULL and the buflen to 0.

To send a message with a data buffer, set \*pBuf to the address of the buffer and buflen to the length of the buffer in bytes.

If pBuf is not NULL and buflen is not 0, HS MSG copies the data buffer into internal buffers. The application can dispose of the buffer (if it is not statically allocated) on return from this function.

The maximum length of the data buffer is 65535 bytes.

This function automatically calls HsMsgTick() in order to deliver the message to destination and for incoming messages to continue to be received into the message queues.

Messages can be sent both to remote destinations located on different hosts or to message queues located on the same host as the sender. In order to send to a local queue, use local loopback address (127.0.0.1), local inbox port in the destination\_address string, for example "127.0.0.1:1234" and specify the desired local queue name in the destination\_queue\_name parameter.

## 2.2 Structures

### 2.2.1 hsmmsg\_message\_t

Message structure for sending and receiving messages

```
#include "hsmmsg_if.h"
```

```
typedef struct
{
    unsigned short msg_code;
    unsigned short from_id;
    unsigned short to_id;
    unsigned long arg1;
    unsigned long arg2;
    unsigned short retcode;
    unsigned short buflen;
    unsigned char *pBuf;
} hsmmsg_message_t;
```

msg\_code – user defined message code

from\_id – user defined from entity id

to\_id – user defined to entity id

arg1 – user defined argument 1

arg2 – user defined argument 2

retcode – user defined return code

buflen – length of buffer in bytes attached to the message

pBuf – pointer to the buffer attached to the message

## 2.3 Definitions

### 2.3.1 Return Codes

```
#include "hsmmsg_if.h"
typedef enum
{
    HSMMSG_RC_OK = 0,
    HSMMSG_RC_INV_PAR = -1,
    HSMMSG_RC_INBOX_NOTALLOC = -2,
    HSMMSG_RC_NO_CTX = -3,
    HSMMSG_RC_NO_MEM = -4,
    HSMMSG_RC_NO_SRCID = -5,
    HSMMSG_RC_NOT_INIT = -6,
    HSMMSG_RC_INBOX_ALLOC = -7,
    HSMMSG_RC_SOCKET_OPEN = -8,
    HSMMSG_RC_SOCKET_BIND = -9,
    HSMMSG_RC_SOCKET_INIT = -10,
    HSMMSG_RC_QNAME_EXISTS = -11,
    HSMMSG_RC_Q_ALLOC = -12,
    HSMMSG_RC_QNOTALLOC = -13,
    HSMMSG_RC_QNOTFOUND = -14,
    HSMMSG_RC_QFULL = -15,
    HSMMSG_RC_QEMPTY = -16,
    HSMMSG_RC_TIMEOUT = -17,
    HSMMSG_RC_EVAL = -18,
} hsmmsg_rc_t;
```

Name	Description
HSMMSG_RC_OK	Success
HSMMSG_RC_INV_PAR	Invalid parameters
HSMMSG_RC_INBOX_NOTALLOC	Inbox not allocated
HSMMSG_RC_NO_CTX	Out of free message transfer contexts
HSMMSG_RC_NO_MEM	Out of memory
HSMMSG_RC_NO_SRCID	Failed to allocated source message session id
HSMMSG_RC_NOT_INIT	Library not initialized
HSMMSG_RC_INBOX_ALLOC	Failed to allocate inbox
HSMMSG_RC_SOCKET_OPEN	Socket layer open failed

HSMSG_RC SOCK_BIND	Socket layer bind failed
HSMSG_RC SOCK_INIT	Socket interface library initialization failed
HSMSG_RC_QNAME_EXISTS	Queue name already exists
HSMSG_RC_Q_ALLOC	Failed to allocate new queue
HSMSG_RC_QNOTALLOC	Queue not allocated
HSMSG_RC_QNOTFOUND	Queue not found
HSMSG_RC_QFULL	Queue full
HSMSG_RC_QEMPTY	Queue empty
HSMSG_RC_TIMEOUT	Timeout
HSMSG_RC_EVAL	Evaluation version exceeded number of messages allowed to send.

## 3 Sample Applications

HS MSG comes with the following sample applications:

- `hsmg_chat`
- `sendfile`
- `recvfile`

The sample applications with source code are located in the folder  
`\SourceCode\SampleApps\`

This section explains how to use the sample applications.

### 3.1 *hsmg\_chat*

#### 3.1.1 Purpose

`hsmg_chat` is a command line sample application that demonstrates:

- HS MSG initialization
- Creating local message inbox and local message queue
- Sending messages to remote inbox / queue
- Receiving messages
- Library de-initialization

#### 3.1.2 Syntax

```
hsmg_chat <local port> <remote ip:port>
```

local port – IP port number to use for creating the local inbox

remote ip: remote IP address where to send messages

port: remote IP port used by the remote destination to create the inbox

#### 3.1.3 Usage

To use the `hsmg_chat` run it on two computers between which you intend to test the message communication.

Example:

```
hsmg_chat 1234 192.168.1.1:1234
```

In this example, both the local sender and the receiver used the same IP port number to create their inboxes.

Type a string at the `>` prompt and press `<ENTER>` - the message is sent to the destination and is displayed by the receiving `hsmg_chat` application.

If the remote `hsmg_chat` application sends a message it is displayed on screen.

To exit, on Windows platform, click `<ESC>`, on linux and mac osx, type `Q<ENTER>`

## **3.2 sendfile**

### **3.2.1 Purpose**

`sendfile` is a command line sample application that demonstrates:

- HS MSG initialization
- Creating local message inbox
- Sending messages to remote inbox / queue
- Library de-initialization

`sendfile` sample sends a file specified on the command line parameter to the sample application `recvfile` that is running on the remote PC.

`Sendfile` uses `HsMsgSendMessage` function to send the next block of the file. `HsMsgSendMessage` does not return until the file block is received by the remote peer. The file block size exceeds the amount of data that can be sent in a single HS MSG protocol unit, so `sendfile` and `recvfile` samples also demonstrate message segmentation and re-assembly

### **3.2.2 Syntax**

`sendfile <remote ip:port> filename`

remote ip: remote IP address where to send messages

port: remote IP port used by the remote destination to create the inbox

filename – name of the file to send

### **3.2.3 Usage**

Use the `sendfile` as follows:

1. On destination host, run `recvfile` sample specifying the local IP port number to use for communication, for example:

```
recvfile 1234
```

2. On local host run `sendfile` sample as specifying the remote address and the filename to send, for example:

```
sendfile 10.1.1.2:1234 myfile.txt
```

The file is being sent to the destination host. The `recvfile` sample saves the file into the working folder of the `recvfile` sample.

The `sendfile` sample exits when the file has been sent or error occurred

## **3.3 *recvfile***

### **3.3.1 Purpose**

recvfile is a command line sample application that demonstrates:

- HS MSG initialization
- Creating local message inbox
- Creating a message queue
- Receiving messages
- Library de-initialization

recvfile sample receives and saves a file send by another sample application sendfile.

### **3.3.2 Syntax**

recvfile <ip port>

ip port: local IP port tp be used when creating the local inbox for receiving messages from the remote peer.

### **3.3.3 Usage**

Use the sendfile as follows:

1. On local host, run recvfile sample specifying the local IP port number to use for communication, for example:

```
recvfile 1234
```

2. On remote host run sendfile sample specifying the remote address and the filename to send, for example:

```
sendfile 10.1.1.2:1234 myfile.txt
```

The file is being received and saves the into the working folder of the recvfile sample.

The recvfile sample exits when the file has been received or error occurred